Juropa3 Experimental Partition

Batch System – SLURM

User's Manual ver 0.2

Apr 2014 @ JSC
Chrysovalantis Paschoulas | c.paschoulas@fz-juelich.de

Contents

1. System Information
2. Modules
3. Slurm Introduction
4. Slurm Configuration
5. Compilers
6. Job Scripts Examples
7. Interactive Jobs
8. Using MICs
9. Using GPUs
10. Examples

1. System Information

Juropa3 is a new test cluster in JSC. Juropa3 is divided into two partitions: the experimental partition and a small partition dedicated to ZEA-1 group. The experimental partition of Juropa3 is going to be used for experiments and testing of new technologies (Hardware and Software) in order to be prepared for the next big installation of Juropa4.

Some of the the technologies and the features that will be used and tested on this partition are:

- Scientific Linux OS (6.4 x86_64): in order to gain experience and move to a RedHat based installation for the next system.
- New Connect-IB Mellanox Cards.
- SLURM as the Batch System: we want a license-free solution for the Batch System and also support for MICs and GPUs.
- End-to-End Data Integrity: this is a new feature of Lustre 2.4 with T-Platforms support.
- Checkpoint-Restart Mechanism for the jobs: T-Platforms will provide libraries and tools for CR using local disks on a set of compute nodes.

Cluster Nodes

For the experimental partition we have 1 Login, 2 Master, 1 Admin and 44 Compute nodes. Also we have 2 Lustre servers and 1 GPFS Gateway. Here is the list with all the nodes of the cluster:

Type (Node Num.)	Hostname	CPU	Cores(VCores)	RAM	Description	Attributes*
Login (1)	juropa3.zam.kfa-juelich.de j3l02	2x Intel Xeon E5-2650 @ 2GHz	16 (32)	128 GB	Login Node	-
Master (1)	juropa3b1.zam.kfa-juelich.de j3b01	2x Intel Xeon E5-2620 @ 2GHz	12 (24)	64 GB	Primary Master Node	-
Master (1)	juropa3b2.zam.kfa-juelich.de j3b02	2x Intel Xeon E5-2620 @ 2GHz	12 (24)	64 GB	Backup Master Node for failover	-
Admin (1)	j3a01	2x Intel Xeon E5-2620 @ 2GHz	12 (24)	64 GB	Admin Node & GPFS Gateway	-
Lustre (2)	j3m[01-02]	2x Intel Xeon E5-2620 @ 2GHz	6 (12)	64 GB	Lustre Servers	-
Compute (28)	j3c[001-0028]	2x Intel Xeon E5-2650 @ 2GHz	16 (32)	64 GB	Disk-less compute nodes	diskless, white
Compute (8)	j3c[031-038]	2x Intel Xeon E5-2650 @ 2GHz	16 (32)	64 GB	Compute nodes with local disks for checkpoint-restart mechanism	cr, ldisk, black
Compute (4)	j3c[053-056]	2x Intel Xeon E5-2650 @ 2GHz	16 (32)	64 GB	Compute nodes with 2x GPUs	gpu, ldisk, yellow
Compute (4)	j3c[057-057]	2x Intel Xeon E5-2650 @ 2GHz	16 (32)	64 GB	Compute nodes with 2x MICs	mic, ldisk, green

- *The attributes are feature names that we gave to the compute nodes for the batch system.*
-

Filesystems

On Juropa3 experimental partition we are providing GPFS and Lustre filesystems. We have home and scratch GPFS filesystems and also an extra Lustre scratch filesystem. Here is a small matrix with all available filesystems to the users:

Type	Mount Points
GPFS WORK	`/work`

GPFS HOME	`/homea`
	`/homeb`
	`/homec`

GPFS ARCH	`/arch`
	`/arch1`
	`/arch2`

User local binaries (GPFS)	`/usr/local`
Lustre WORK	`/lustre/work`

Access to the Cluster

Users can connect to the login node with the ssh command:

```
> ssh <username>@juropa3.zam.kfa-juelich.de
```

2. Modules

All the available software on the cluster (compilers, tools, libraries, etc..) is provided in the form of modules. The user in order to use the desired software they have to use the module command. With this command the user can load or unload the software or a specific version of the required software. By default some modules are preloaded for all users.

Here is a list of useful options:

Command	**Description**
module list	Print a list with all the currently loaded modules
module avail	Display all available modules
module load <module name>	Load a module
module unload <module name>	Unload a module
module purge	Unload all currently loaded modules

Default Packages

The default packages for the users are the Intel Compiler and the Parastation MPI:

```
1) parastation/mpi2-intel-5.0.27-1
2) intel/13.1.0
```

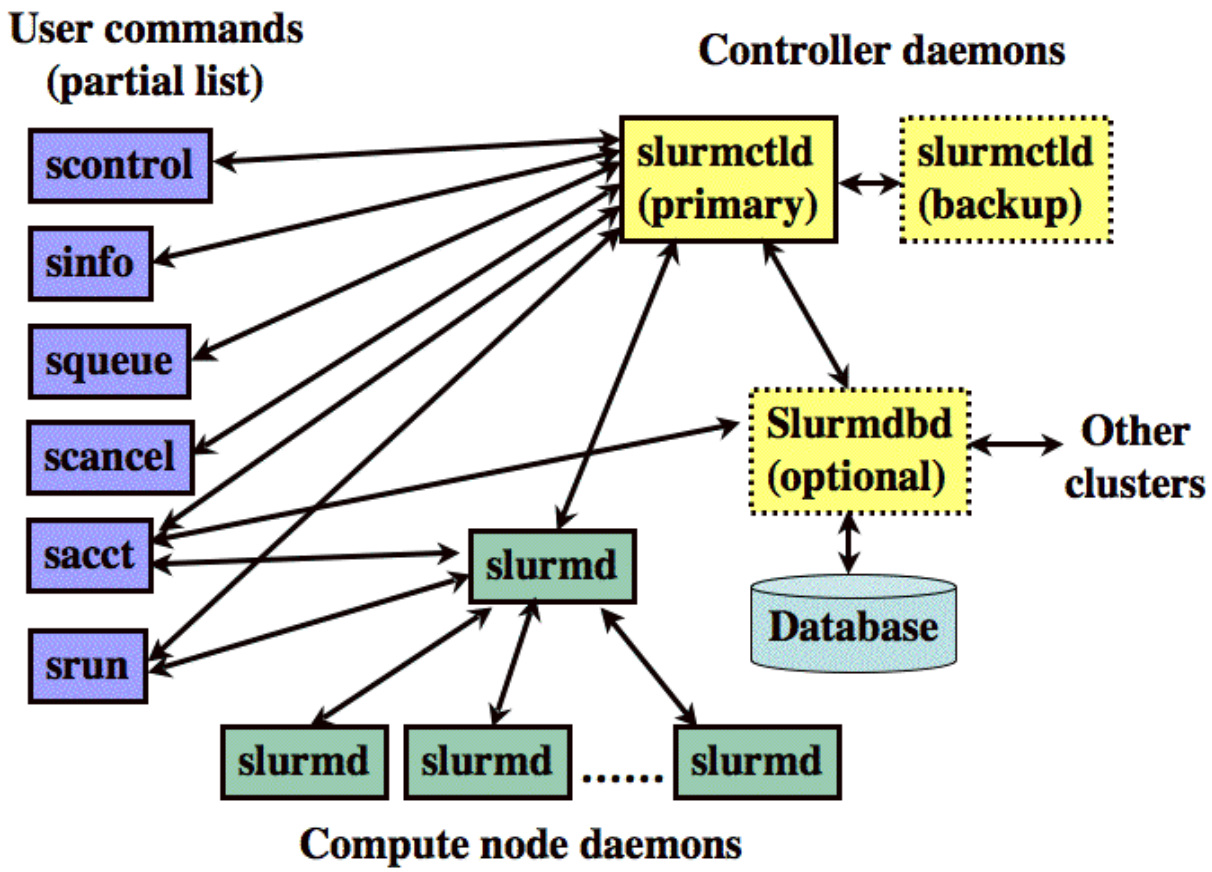
Examples

```
[user@j3l02 jobs]$ module list
Currently Loaded Modulefiles:
 1) parastation/mpi2-intel-5.0.27-1  2) intel/13.1.0

[user@j3l02 jobs]$ module purge

[user@j3l02 jobs]$ module load intel impi

[user@j3l02 jobs]$ module list
Currently Loaded Modulefiles:
 1) intel/13.1.0    2) impi/4.0.3.008

[user@j3l02 jobs]$ module avail intel
------------ /usr/local/modulefiles/COMPILER ------------
intel/11.0            intel/12.0.4          intel/12.1.2
intel/11.1.059        intel/12.0.5          intel/12.1.4
intel/11.1.072        intel/12.1.0          intel/13.1.0(default)
intel/12.0.3          intel/12.1.1
------------ /usr/local/modulefiles/MATH ------------
------------ /usr/local/modulefiles/SCIENTIFIC ------------
------------ /usr/local/modulefiles/IO ------------
------------ /usr/local/modulefiles/TOOLS ------------
------------ /usr/local/modulefiles/MISC ------------
```

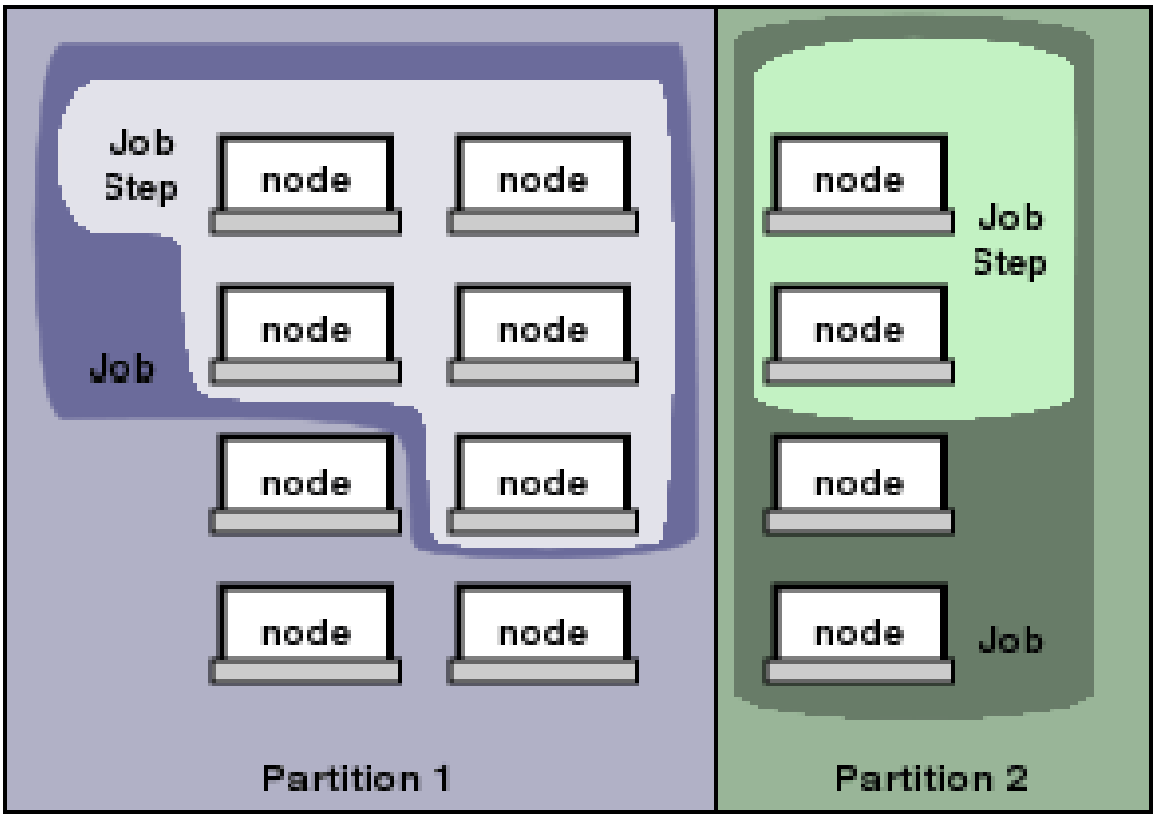
3. Slurm Introduction

The Simple Linux Utility for Resource Management (SLURM) is an open source, fault-tolerant, and highly scalable cluster management and job scheduling system for large and small Linux clusters. SLURM requires no kernel modifications for its operation and is relatively self-contained. As a cluster resource manager, SLURM has three key functions. First, it allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work. Second, it provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes. Finally, it arbitrates contention for resources by managing a queue of pending work.

SLURM consists of a **slurmd** daemon running on each compute node and a central **slurmctld** daemon running on a management node (with optional fail-over twin). The **slurmd** daemons provide fault-tolerant hierarchical communications. The user commands include: **sacct**, **salloc**, **sattach**, **sbatch**, **sbcast**, **scancel**, **scontrol**, **sinfo**, **smap**, **squeue**, **srun**, **strigger** and **sview**. All of the commands can run anywhere in the cluster (job submission is allowed only on the login node “j3l02”).



The entities managed by these SLURM daemons include **nodes**, the compute resource in SLURM, **partitions**, which group nodes into logical (possibly overlapping) sets, **jobs**, or allocations of resources assigned to a user for a specified amount of time, and **job steps**, which are sets of (possibly parallel) tasks within a job (**srun** starts a **job step** using a subset or all compute nodes of the allocated nodes for the job). The partitions can be considered job queues, each of which has an assortment of constraints such as job size limit, job time limit, users permitted to use it, etc. Priority-ordered jobs are allocated nodes within a partition until the resources (nodes, processors, memory, etc.) within that partition are exhausted. Once a job is assigned a set of nodes, the user is able to initiate parallel work in the form of job steps in any configuration within the allocation. For instance, a single job step may be started that utilizes all nodes allocated to the job, or several job steps may independently use a portion of the allocation.



List of Commands

Man pages exist for all SLURM daemons, commands, and API functions. The command option --help also provides a brief summary of options. Note that the command options are all case insensitive.

sacct is used to report job or job step accounting information about active or completed jobs.

salloc is used to allocate resources for a job in real time. Typically this is used to allocate resources and spawn a shell. The shell is then used to execute srun commands to launch parallel tasks.

sattach is used to attach standard input, output, and error plus signal capabilities to a currently running job or job step. One can attach to and detach from jobs multiple times.

sbatch is used to submit a job script for later execution. The script will typically contain one or more srun commands to launch parallel tasks.

sbcast is used to transfer a file from local disk to local disk on the nodes allocated to a job. This can be used to effectively use diskless compute nodes or provide improved performance relative to a shared file system.

scancel is used to cancel a pending or running job or job step. It can also be used to send an arbitrary signal to all processes associated with a running job or job step.

scontrol is the administrative tool used to view and/or modify SLURM state. Note that many scontrol commands can only be executed as user root.

sinfo reports the state of partitions and nodes managed by SLURM. It has a wide variety of filtering, sorting, and formatting options.

smap reports state information for jobs, partitions, and nodes managed by SLURM, but graphically displays the information to reflect network topology.

sprio shows the priorities of queued jobs.

squeue reports the state of jobs or job steps. It has a wide variety of filtering, sorting, and formatting options. By default, it reports the running jobs in priority order and then the pending jobs in priority order.

srun is used to submit a job for execution or initiate job steps in real time. srun has a wide variety of options to specify resource requirements, including: minimum and maximum node count, processor count, specific nodes to use or not use, and specific node characteristics (so much memory, disk space, certain required features, etc.). A job can contain multiple job steps executing sequentially or in parallel on independent or shared nodes within the job's node allocation.

sstat gives various status information of a running job/step.

strigger is used to set, get or view event triggers. Event triggers include things such as nodes going down or jobs approaching their time limit.

sview is a graphical user interface to get and update state information for jobs, partitions, and nodes managed by SLURM.

4. Slurm Configuration

The current Slurm configuration is not the final. We will continuously keep working on Slurm testing some feauters until we reach the desired configuration.

Current Configuration

Control servers: slurmctld on j3b01 + backup controller on j3b02 for HA
Scheduler: backfill
Accounting: advanced accounting using slurmdbd with MySQL + backup daemon
Priorities: multifactor priorities policy
Preemption: NO
HW Support: GPUs & MICs support (MICs in Native Mode also)

Queues

The partition configuration permits you to establish different job limits or access controls for various groups (or partitions) of nodes. Nodes may be in more than one partition, making **partitions serve as general purpose queues**. For example one may put the same set of nodes into two different partitions, each with different constraints (time limit, job sizes, groups allowed to use the partition, etc.). Jobs are allocated resources within a single partition.

The configured partitions on Juropa3 are:

Partition Name	**Node List**	**Description**
batch	`j3c[001-028,031-038,057-060]`	Default queue, all compute nodes are included
q_diskless	`j3c[001-028]`	Diskless compute nodes
q_cr	`j3c[031-038]`	Diskless compute nodes, with local disks used only by the Checkpoint-Restart mechanism
q_gpus	`j3c[053-056]`	Compute nodes with GPUs (not in batch queue!)
q_mics	`j3c[057-060]`	Compute nodes with MICs
maint	`ALL`	Special queue for the admins

5. Compilers

On Juropa3 ZEA-1 partition we offer some wrappers to the users, in order to compile and execute parallel jobs using MPI (like on Juropa2). We provide different wrappers depending on the MPI version that is used. Users can choose the compiler's version using the module command.

ParaStation MPI

The available wrappers for Parastation MPI are:

mpicc, mpicxx, mpif77, mpif90

To execute a parallel application it is recommended to use the ***mpiexec*** command.

Intel MPI

The available wrappers for Intel MPI are:

mpiicc, mpiicpc, mpiifort

To execute a parallel application it is recommended to use the ***srun*** command.

Compiler options

`-openmp`	enables OpenMP
`-g`	creates debugging information
`-L`	path to libraries for the linker
`-O[0-3]`	optimization levels

Compilation examples

a) MPI program in C++:

```
> mpicxx -O2 program.cpp -o mpi_program
```

b) Hybrid MPI/OpenMP program in C:

```
> mpicc -openmp -o exe_program code_program.c
```

6. Job Scripts Examples

Users can submit jobs using the sbatch command. In the job scripts, in order to define the sbatch parameters you have to use the #SBATCH directives. Users can also start jobs using directly the srun command. But the best way to submit a job is to use sbatch in order to allocate the required resource with the desired walltime and then call mpiexec or srun inside the script. With srun users can create jobs steps. A job step can allocate the whole or a subset of the already allocated resources from sbatch. So with these commands Slurm offers a mechanism to allocate resources for a certain walltime and then run many parallel jobs in that frame.

Non-parallel job

Here is a simple example where we execute 2 system commands inside the script, sleep and hostname. This job will have a name as TestJob, we allocated 1 compute node, we defined the output files and we requested 30 minutes walltime.

```
#!/bin/bash
#SBATCH -J TestJob
#SBATCH -N 1
#SBATCH -o TestJob-%j.out
#SBATCH -e TestJob-%j.err
#SBATCH --time=30

sleep 5

hostname
```

We could do the same using directly the srun command (accepts only one executable as argument):

```
> srun -N1 –-time=30 hostname
```

Parastation MPI

A SPANK plugin was implemented for Slurm in order to communicate correctly with the Parastation environment and its MPI implementation. To start a parallel job using Parastation MPI users have to use the mpiexec command.

In the following example we have an MPI application that will start 1024 MPI tasks on 32 nodes with 32 taks per node. The walltime is one hour.

```
#!/bin/bash
#SBATCH -J TestJob
#SBATCH -N 32
#SBATCH -n 1024
#SBATCH --ntasks-per-node=32
#SBATCH –time=60

mpiexec -np 1024 ./mpiexe
```

In the example we have a hybrid MPI/OpenMP job. We allocate 5 compute nodes for 2 hours. The job will have 40 MPI tasks in total, 8 tasks per node and 4 OpenMP threads per task. Important is to define the env variable OMP_NUM_THREADS.

```
#!/bin/bash
#SBATCH -J TestJob
#SBATCH -N 5
#SBATCH -n 40
#SBATCH --ntasks-per-node=8
#SBATCH --cpus-per-task=4
#SBATCH -o TestJob-%j.out
#SBATCH -e TestJob-%j.err
#SBATCH --time= 02:00:00

export OMP_NUM_THREADS=4

mpiexec -np 40 ./hybrid exe
```

Intel MPI

In order to use Intel MPI users have to unload Parastation MPI and load the module for Inte MPI. Also the users have to export some environment variables in order to make Intel MPI work properly. The list of these variables is:

```
I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
DAT_OVERRIDE=/etc/rdma/dat.conf
```

The users have to export also some variables for the communication between the MPI tasks. There are two options with the same performance:

```
I_MPI_DEVICE=rdma
I_MPI_FABRICS=dapl
```

or just

```
I_MPI_FABRICS=ofa
```

If the users want some extra debugging info the have to export:

```
I_MPI_DEBUG=5
```

Here is an example of a job script that uses Intel MPI:

```
#!/bin/bash
#SBATCH -J TestJobIMPI
#SBATCH -N 4
#SBATCH --ntasks-per-node=4
#SBATCH –-time=00:50:00

export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
export DAT_OVERRIDE=/etc/rdma/dat.conf
export I_MPI_FABRICS=ofa
export I_MPI_DEBUG=5

srun -n16 ./testimpi
```

7. Interactive Jobs

To run interactive jobs, users can call srun with some specific arguments. For example:

```
srun -N2 -time=120 --pty -u bash -i -l
```

This command will return a console from the compute nodes of the compute nodes. Every command that will called there it will be executed on all allocated compute nodes.

```
Login node:
[paschoul@j3l02 jobs]$ srun -N2 -time=120 --pty -u bash -i -l

Compute node: [Allocated 2 nodes: j3c001 and j3c002]
[paschoul@j3c001 jobs]$ srun -N2 hostname
j3c001
j3c002

[paschoul@j3c001 jobs]$ srun -N1 -n1 hostname
j3c001
...
```

Another way to start an interactive job is to call salloc. Please choose the way you like more.

8. Using MICs

Currently the MICS can be used only in offload mode. In this part we have documentation about how users can compile and run MIC code in both cases of a) Offload mode and b) Intel MPI+Offload mode.

Offload Mode

Here is an example of source code that will run on MICs, file “hello_offload.c”:

```
#include <stdio.h>
#include <stdlib.h>

void print_hello_host()
{
//"Hello from Host" on the host.

  printf( "Hello from HOST!\n" );

  return;
}

__attribute__ ((target(mic)))
void print_hello_mic()
{
//"Hello from Phi" on the coprocessor.

  printf( "Hello from Phi!\n" );

  return;
}

int main( int argc, char** argv )
{
// Hello function is called on the host.
    print_hello_host();

// Below you may choose on which mic you want your function to run.
    #pragma offload target (mic:0)
//     #pragma offload target (mic:1)
    print_hello_mic();

    return 0;
}
```

To compile:

```
> icc -O3 -g  hello_offload.c -o hello_offload.exe
```

The job script “offload.sh”:

```
#!/bin/bash
#SBATCH -N 1
#SBATCH -p queue_mics
#SBATCH –time=30

# The next 2 variables can be used in order to be sure
# that your code was offloaded and run on the MIC and
# on which MIC. Possible values range between 0-3.
```

```
export H_TRACE=1
export OFFLOAD_REPORT=1

./hello_offload.exe
```

To submit:

```
> sbatch offload.sh
```

The results will be given on the "slurm-<batchJobID>.out"

MPI + Offload Mode

Here is the source code, file “hello_mpi_offload.c”:

```
#include <stdio.h>
#include <stdlib.h>
#include <mpi.h>

void print_hello_host()
{
//"Hello from Host" on the host.
  printf( "Hello from HOST!\n" );
  return;
}

__attribute__ ((target(mic)))
void print_hello_mic()
{
//"Hello from Phi" on the coprocessor.
  printf( "Hello from Phi!\n" );
  return;
}

int main( int argc, char** argv )
{
  int rank, size;
  char hostname[255];

  MPI_Init(&argc,&argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  gethostname(hostname,255);
  printf("Hello from process %d of %d on %s\n", rank, size, hostname);

// Hello function is called on the host.
  print_hello_host();

// The same function shall be called in an offload region.
// Below we choose the function to run firstly on MIC0 and then on MIC1.
  #pragma offload target (mic:0)
  print_hello_mic();

  #pragma offload target (mic:1)
  print_hello_mic();


  MPI_Finalize();
  return 0;
}
```

There are two possible ways to compile and run the executable. The combinations that the users may use are Parastation/MPI(mpicc)+srun and Intel/MPI(mpiicc)+srun. In the first way we noticed that the task creation is buggy, because it creates MPI tasks with different MPI COMM_WORLDs. So the users are advised to use the second way with Intel MPI.

So the users have to load the Intel MPI module first:

```
> module purge
> module load intel
> module load impi
```

Compile options:

```
> mpiicc -O3 -g  hello_mpi_offload.c -o hello_mpi_offload.exe
```

Job script “mpi_offload.sh”:

```
#!/bin/bash

#SBATCH -N 2
#SBATCH --ntasks-per-node=2
#SBATCH -p queue_mics
#SBATCH --time=30

export I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so
export I_MPI_DEVICE=rdma
export DAT_OVERRIDE=/etc/rdma/dat.conf
export I_MPI_FABRICS=dapl
# export I_MPI_DEBUG=5

# The next 2 variables can be used in order to be sure
# that your code was offloaded and run on the MIC and
# on which MIC. Possible values range between 0-3.
export H_TRACE=1
export OFFLOAD_REPORT=1

srun -n 4 ./hello_mpi_offload.exe
```

Job submission:

```
> sbatch mpi_offload.sh
```

9. Using GPUs

Coming soon...

!TODO!

*** FYI: We have 4 compute nodes j3c[053-056] with GPUs installed on them. Each node has 2 NVIDIA Tesla K20X.

10. Examples

Job submission

```
$ sbatch <jobscript>
```

Check all queues

```
$ sinfo
PARTITION      AVAIL  TIMELIMIT  NODES  STATE NODELIST
batch*            up 1-00:00:00      1  drain j3c053
batch*            up 1-00:00:00      2  alloc j3c[002-003]
batch*            up 1-00:00:00     41   idle j3c[001,004-028,031-038,054-060]
queue_diskless    up 1-00:00:00      2  alloc j3c[002-003]
queue_diskless    up 1-00:00:00     26   idle j3c[001,004-028]
queue_cr          up 1-00:00:00      8   idle j3c[031-038]
queue_normal      up 1-00:00:00      2  alloc j3c[002-003]
queue_normal      up 1-00:00:00     34   idle j3c[001,004-028,031-038]
queue_gpus        up 1-00:00:00      1  drain j3c053
queue_gpus        up 1-00:00:00      3   idle j3c[054-056]
queue_mics        up 1-00:00:00      4   idle j3c[057-060]
```

Check a certain queue

```
$ sinfo -p queue_diskless
PARTITION      AVAIL  TIMELIMIT  NODES  STATE NODELIST
queue_diskless    up 1-00:00:00      2  alloc j3c[002-003]
queue_diskless    up 1-00:00:00     26   idle j3c[001,004-028]
```

Check all jobs in the queue

```
$ squeue
             JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
              1331     batch     bash paschoul   R    1:02:15      2 j3c[002-003]
```

Check all jobs of a user

```
$ squeue -u paschoul
             JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
              1331     batch     bash paschoul   R    1:13:04      2 j3c[002-003]
```

Get information about all jobs

```
$ scontrol show job
JobId=1331 Name=bash
…
```

Get information about one job

```
$ scontrol show job 1342
JobId=1342 Name=mytest5
   UserId=lguest(1006) GroupId=lguest(1006)
   Priority=4294901739 Account=(null) QOS=(null)
   JobState=COMPLETED Reason=None Dependency=(null)
   Requeue=0 Restarts=0 BatchFlag=1 ExitCode=0:0
   RunTime=00:00:01 TimeLimit=06:00:00 TimeMin=N/A
   SubmitTime=2013-07-31T12:47:57 EligibleTime=2013-07-31T12:47:57
   StartTime=2013-07-31T12:47:57 EndTime=2013-07-31T12:47:58
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   Partition=batch AllocNode:Sid=j3l02:12699
   ReqNodeList=(null) ExcNodeList=(null)
   NodeList=j3c[004-008]
   BatchHost=j3c004
   NumNodes=5 NumCPUs=160 CPUs/Task=1 ReqS:C:T=*:*:*
   MinCPUsNode=1 MinMemoryNode=0 MinTmpDiskNode=0
   Features=(null) Gres=(null) Reservation=(null)
   Shared=0 Contiguous=0 Licenses=(null) Network=(null)
   ...
```

Check the configuration and state of all nodes

```
$ scontrol show node
NodeName=j3c001 Arch=x86_64 CoresPerSocket=8
   CPUAlloc=0 CPUErr=0 CPUTot=32 CPULoad=0.00 Features=diskless,white
   Gres=(null)
   NodeAddr=j3c001 NodeHostName=j3c001
   OS=Linux RealMemory=64534 AllocMem=0 Sockets=2 Boards=1
   State=IDLE ThreadsPerCore=2 TmpDisk=0 Weight=1
   BootTime=2013-07-12T13:16:22 SlurmdStartTime=2013-07-31T10:04:49
   CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s


NodeName=j3c002 Arch=x86_64 CoresPerSocket=8
   CPUAlloc=32 CPUErr=0 CPUTot=32 CPULoad=0.00 Features=diskless,white
   Gres=(null)
   NodeAddr=j3c002 NodeHostName=j3c002
   OS=Linux RealMemory=64534 AllocMem=0 Sockets=2 Boards=1
   State=ALLOCATED ThreadsPerCore=2 TmpDisk=0 Weight=1
   BootTime=2013-07-12T13:22:42 SlurmdStartTime=2013-07-31T10:04:49
   CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s

...
```

Check the configuration and state of one node

```
$ scontrol show node j3c004
NodeName=j3c004 Arch=x86_64 CoresPerSocket=8
   CPUAlloc=0 CPUErr=0 CPUTot=32 CPULoad=0.00 Features=diskless,white
   Gres=(null)
   NodeAddr=j3c004 NodeHostName=j3c004
   OS=Linux RealMemory=64534 AllocMem=0 Sockets=2 Boards=1
   State=IDLE ThreadsPerCore=2 TmpDisk=0 Weight=1
   BootTime=2013-07-12T14:14:26 SlurmdStartTime=2013-07-31T10:04:49
   CurrentWatts=0 LowestJoules=0 ConsumedJoules=0
   ExtSensorsJoules=n/s ExtSensorsWatts=0 ExtSensorsTemp=n/s
```

Check the configuration and state of all partitions

```
$ scontrol show partition
PartitionName=batch
   AllocNodes=j3l02 AllowGroups=ALL Default=YES
   DefaultTime=06:00:00 DisableRootJobs=YES GraceTime=0 Hidden=NO
   MaxNodes=44 MaxTime=1-00:00:00 MinNodes=1 MaxCPUsPerNode=UNLIMITED
   Nodes=j3c0[01-28],j3c0[31-38],j3c0[53-56],j3c0[57-60]
   Priority=1 RootOnly=NO ReqResv=NO Shared=NO PreemptMode=OFF
   State=UP TotalCPUs=1408 TotalNodes=44 SelectTypeParameters=N/A
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED

PartitionName=queue_diskless
   AllocNodes=j3l02 AllowGroups=ALL Alternate=batch Default=NO
   DefaultTime=06:00:00 DisableRootJobs=YES GraceTime=0 Hidden=NO
   MaxNodes=44 MaxTime=1-00:00:00 MinNodes=1 MaxCPUsPerNode=UNLIMITED
   Nodes=j3c0[01-28]
   Priority=1 RootOnly=NO ReqResv=NO Shared=NO PreemptMode=OFF
   State=UP TotalCPUs=896 TotalNodes=28 SelectTypeParameters=N/A
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED

...
```

Check the configuration and state of one specific partition

```
$ scontrol show partition queue_diskless
PartitionName=queue_diskless
   AllocNodes=j3l02 AllowGroups=ALL Alternate=batch Default=NO
   DefaultTime=06:00:00 DisableRootJobs=YES GraceTime=0 Hidden=NO
   MaxNodes=44 MaxTime=1-00:00:00 MinNodes=1 MaxCPUsPerNode=UNLIMITED
   Nodes=j3c0[01-28]
   Priority=1 RootOnly=NO ReqResv=NO Shared=NO PreemptMode=OFF
   State=UP TotalCPUs=896 TotalNodes=28 SelectTypeParameters=N/A
   DefMemPerNode=UNLIMITED MaxMemPerNode=UNLIMITED
```

Cancel a job

```
$ squeue
             JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
              1331     batch     bash paschoul   R    1:17:08      2 j3c[002-003]

$ scancel 1331
```

Hold a job that is in queue but not running

```
$ hold 1331
```

Release a job from hold

```
$ release 1331
```